CPE 349: Lab 2-1: Permutations

Deliverable:

Source code for a single class, **CombObjects.java** with the methods described below. Submit on PolyLearn.

Generating combinatorial objects is fundamental to many problems in computer science. In this assignment you will implement algorithms to: 1. generate the permutations of a set of lower case letters in **lexicographic order**. 2. Generate permutations in an order where the adjacent permutations differ only in the exchange of two adjacent entries.

Your Class, **CombObjects.java**, must meet the following specifications:

- Implement a method **getLexPerm (String str)** that returns an ArrayList of Strings in lexicographic order where each string represents a permutation of the character in **str**. You may assume the input string represents a set of distinct letters in order, e.g. *abcd represents {a, b, c, d}*.
- Implement a method **getMinChgPerm (String str)** that returns an ArrayList of Strings that satisfy a minimum change requirement. Again the input argument can be assumed to be a string of distinct lower case letters (in alphabetical order). You may assume the input is correct and represents a set of distinct letters in order, e.g. *abcd represents {a, b, c, d}*.
- Your program must be well structured, commented, and easy to read.
- Both methods must be **recursive and must follow the high level description below**. See below for the desired output for lists of permutations for inputs "abc" and "abcd" of strings with 3 or 4 elements. You must match the results exactly.

Description of recursive algorithm to generate permutations in lexicographic order

```
//  Assumes string contains characters in appropriate order
If the string is contains a single character return a string containing the single
character in a ArrayList    // this is a possible base case although you could also
//use the string being empty as the base case; return a list containing the empty string

Loop through all character positions of the string containing the characters to be
permuted, for each character
      Form a simpler word by removing the character
      Generate all permutations of the simpler word recursively
      Add the removed character to the front of each permutation of the simpler word, and
            add the resulting permutation to a list
Return these newly constructed permutations
```

Description of recursive algorithm to generate permutations satisfying the minimum change requirement

```
//  Assumes string contains characters in appropriate order
If the string is empty return it
Remove the last character, call it x, of the string
Generate all permutations (satisfying min change requirement) of the simpler word
Loop over the returned permutations
```

- insert the removed character into a returned permutation into all possible positions moving right to left
- insert the removed character into the next returned permutation into all possible positions moving left to right

```
Return all these newly constructed permutations
```

Example input for getLexPerm: abc

Output:

```
abc
acb
bac
bca
cab
cba
```

Example input for getLexPerm: abcd

Output:

```
abcd
abdc
acbd
acdb
adbc
adcb
bacd
badc
bcad
bcda
bdac
bdca
cabd
cadb
cbad
cbda
cdab
cdba
dabc
dacb
dbac
dbca
dcab
dcba
```

Example input for getMinChgPerm: abc

Output:

```
abc
acb
cab
cba
bca
bac
```

Example input for getMinChgPerm: abcd

Output:

```
abcd
abdc
adbc
dabc
dacb
adcb
acdb
acbd
cabd
cadb
cdab
dcab
dcba
cdba
cbda
cbad
bcad
bcda
bdca
dbca
dbac
bdac
badc
bacd
```